Creating a Codio account
Codio is a web-based web development platform that makes it simple and easy to work on web projects and share them with collaborators.

1. Go to http://codio.com
2. Click sign up. Sign up with github or just fill out the necessary info.
3. You'll then be presented with your dashboard, where you'll collect all your future codio projects.

Creating a project
Codio allows you to work with a number of programming languages in a single environment. For the purposes of this tutorial, we will be focusing on the critical parts of web development: using HTML and CSS to create standard web pages.

4. Select 'Create Project' from the dashboard to create a new project.
5. Under 'Project Name', choose a name; such as "Tutorial" or "My First Website".
6. Select 'Public: Open and viewable to all'. We'll keep these tutorials open-source so if you put together a cool feature, feel free to share it with all of us or any of your nerd friends.
7. Under the 'Template' tab, choose 'Empty project' as the selected template for a clean slate.*
8. Select create project.
9. This brings you to the *IDE* (Integrated Development Environment).

*The 'HTML5 Boilerplate' Template option provides some nice backwards-compatible support for older browsers while integrating new HTML5 feature access, but it's a little messy and could be confusing at first.

Working within the IDE
An IDE (Integrated Development Environment) is a platform that aims to provide the right tools for development. The Codio IDE is focused on web development, allowing you to use an array of web development languages and paradigms to make great web apps and sites. A great codio feature is that it allows you to instantly see what your webpage will look like as you develop it. Another great codio feature is its use of panels. The 'View' tab at the top allows you to customize the codio layout however you like it, and even share your great coding layout with your collaborators. Codio also has real-time google-doc style editing between those working on a project together. At left we have the 'Filetree' panel, representing all of the files associated with your project. You may observe that there is a simple *index* page 'index.html' and a *readme* file 'README.md' used to describe your project. The index page is going to be a standard element of most of the websites you'll be creating and working on--think of it as just the main page.
In the center of the screen we have the *workspace*. This is where your working web pages will be displayed, and where you will write your code for the pages.

Above the Filetree and workspace is the *toolbar*, which provides many useful tools.

10. Click 'index.html' to bring up the HTML file to edit. This is all the code used to describe your web page.
11. From the toolbar, select preview. This is your web page as it would be seen on the internet.
12. From the toolbar, click view > layouts > 2 columns. This brings up 2 panels in the workspace so we can see our code and web page side by side as we work.
13. Click 'index.html' from the filetree to bring it up on a panel.
14. Click preview to bring the web page into the workspace. Drag the 'index.html (Preview)' tab to the other panel so you can see both the index file code and its preview on the same page. Now we're ready to code.

HTML structure
HTML (HyperText Markup Language) is the language used to represent the majority of pages on the internet. It is made up of pairs of open and close *tags* (<tag> </tag>) used to represent the different pieces of a page. You may observe that an HTML page begins and ends with an <html> tag (<html>...</html>). The <head> tag denotes the headers of your page, which aren't immediately visible, and the <body> tag denotes the body of your document, which contains the contents of the page that is viewed. Within the body tags, paragraph tags (<p>) denote where a section of text will go relative to other sections on the page. Tags within tags are considered to be *nested*, naturally forming a hierarchy of the elements that make up a webpage. For the full list of available tags check out http://www.w3schools.com/tags/default.asp.
Lets change some things up by first adding a header.

15. In the left panel with your index page code, start a new line after the <body> tag on line 6.
16. Add a new <h1> (header) tag. Upon finishing typing the last '>', codio will *automagically* generate the close tag (</h1>) and place the cursor within the tag for you to enter its contents.
17. Type something along the lines of "*your_name*'s project" *, inserting your name where specified.
18. On the right workspace window, click the refresh symbol (looks kinda like ⟳). This will reload the page to now include any changes you've made. From now on, when I mention refreshing, I am referring to this action.

*The amount of *whitespace* (spaces and tabs) between tags is unimportant and will not make a difference in your site's layout.

HTML elements and attributes
Let's go into a little more detail about tags with HTML. Each open/close pair represents an *element* on an HTML page. Each of these elements have certain attributes that let you modify the look or behavior of the element. Attributes are specified within the tag itself. As an example, we'll add a useless button.

19. After the </p> tag, add a <button> tag.
20. Between the <button> and </button> you specify text to describe the button on the page. Try something like ‘yolo’.
21. Refresh the page on the right and voila, we’ve got a button that does nothing.
22. Within the <button> tag, add an id* attribute to describe our button in code. Within the button’s open tag, add ‘id=”theButton”’, so the final element will look like:
    What’s id for?
    What are classes?
    <button id=”theButton”>yolo</button>

23. Refresh the preview and notice that this attribute has no effect on the actual look of the button.
24. Let’s disable this stupid button so people can’t click on it. In the open tag, add the ‘disabled’ attribute**, so the element looks like <button id=”theButton” disabled>yolo</button>.
25. Refresh and notice that the button is now disabled. Some attributes modify how we as the developer can use the button, and others modify how our users can use the button.

*This id attribute is used later when we talk about styles
**Some attributes act alone, while others come in the form of name=”value” pairs.

Styling with CSS
CSS (Cascading Style Sheets) are used to specifically describe how elements on our pages are styled (red background, bold text, etc). Styling can be done in-line to specific elements with the ‘style’ attribute, but doing a lot of styling in this manner makes your code hard to read and can make it hard to make changes to old code when you want to move forward with a project. For this reason, we choose to *encapsulate* our styling with a separate CSS fileor tag dedicated to style. CSS can be described as an HTML <style> tag at the top of the HTML file. CSS files are often written as their own files, but for small projects like this it makes more sense to just include the tag on the index file.
CSS lets you style elements, ids and classes. We’ll cover the latter two later, but for now lets style some of your page. You may reference all styles options at http://www.w3schools.com/cssref/default.asp.

26. Before the <html> tag, include a <style> tag, along with the usual </style> close tag.
27. Let’s change all paragraph text. Within the style tags, insert “p{}”. This is essentially saying “take all paragraph elements and change them how I want within the curly braces”.
28. Within the curly braces {}, include the styling “color:blue;”*, such that, within the style tags we have something along the lines of “p{color:blue;}”.
29. Refresh and notice that the “Hello World!” text is now blue.

*style is always represented in the form of <property>:<value>; (note the colon and semicolon)

IDs

We don't always want to change the style of an entire element type, so with html id's we can select certain elements to modify. Id's are referenced by CSS with # (pound symbol/hashtag).
Adding styling will be the same as adding styling to paragraphs, but we'll refer only to the id.

30. After the paragraph styling, add styling for our "theButton" id. This will look like "#theButton{}".
31. Style the background color to be yellow. The final result should be something like "#theButton{background-color:yellow;}".
32. Refresh the page. Check out those great colors!

Classes

So we know how to change the styles of full sets of elements, as well as the styles tied to specific id's. If we want to change certain groups of objects we can assign them to *classes*. Classes are much like id's, but have more flexibility because an element can be assigned to multiple classes. Lets make two classes, and see how they affect the look of our site.
what are classes?
In CSS, we denote id's with the hashtag, and we denote classes with the dot (.).

33. In the style tag, after the id styling, add a class "greenTXT" to make our text green. The new class will look like ".greenTXT{}".
34. Include the styling of the class to change the color like we did to the paragraph. The end result will be ".greenTXT{color:#19FF19;}"*
35. Do the same to make a class "blueBG" to provide a blue background. You should include something along the lines of ".blueBG{background-color:#458FFF;}"*
36. Add these classes to your html elements with the class attribute. The value of the attribute should be the class name. For example, we would give the header both classes (separated by a space) by adding ' class="greenTXT blueBG" ' in the open tag.

go into more detail here about header

37. Give these classes to the header, paragraph, and button such that their open tags now look like <h1 class="greenTXT blueBG">, <p class="greenTXT blueBG">, and <button id="theButton" class="greenTXT blueBG" disabled>.
38. Refresh and notice the following:
    a. Your web page now looks sweet
    b. The header had no previous styling associated with is, so it has greenTXT and a blueBG.
    c. The paragraph got the blueBG, but its text color changed from blue to green. This is because styling with classes takes precedence over styling an element type when a class is specifically assigned to an element.
    d. The button's text is now green, but the background stayed yellow. This is because styling by id takes precedence over styling by class or element.

39. Add the same ‘ class=”greenTXT blueBG” ‘ attribute to our <body> element.

*Colors can also be more specifically defined by their hexadecimal values. #19FF19 and #458FFF represent Kappa Theta Pi’s official green and blue, respectively.

Nesting
Classes, Styles, and certain Element attributes are given to all html elements nested within other html elements. Any element within the open and close tags of another element is considered to be nested. An example of this is that the <head> and <body> tags are always nested within the <html> tag, and our <p>, <h1>, and <button> elements are nested within the <body> tag. Let’s experiment a little with nesting our elements.

40. Remove the class attribute from your paragraph, header, and button.
41. Add the class attribute to the body
42. Refresh and observe that the green text and blue background is now applied across the full body of the page. Also note that the paragraph color returns to blue because the class is not directly applied to the element.
43. This is way too much blue though, so lets nest our elements within <div>*, or a vertical divider.
    a. Remove the class attribute from the body tag.
    b. After the body tag, add a divider with our classes: <div class=”greenTXT blueBG”>
    c. Since we want to nest all of our elements, remove the close tag that was created by codio.
    d. Place a </div> close tag right before the </body> close tag.
44. We’ve now nested our header, paragraph, and button within a divider, which is nested in the body, which is nested in the html tag. Refresh and observe that the class styling only applies to the divider.

*dividers are a convenient way to break your web page into specific sections.

Working with multiple files
<link rel="stylesheet" type="text/css" href="theme.css">